Week 10: Programming (Part 2 - Obstacles)

Welcome to Week 10!

Introduction to Programming Obstacles

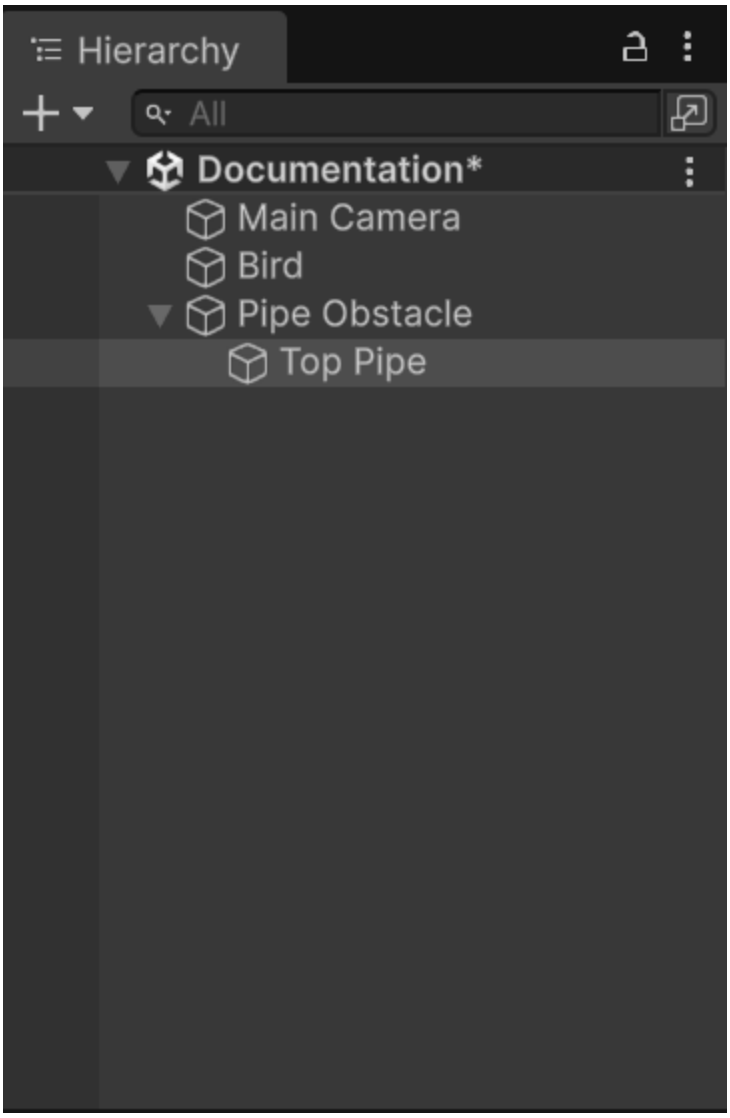
10.1 – Introduction

This week continues the development of the final Unity project, which was started last week. The focus shifts from player control to level challenge by introducing obstacles for the bird to navigate. Specifically, vertical pipe obstacles will be created, animated, spawned at timed intervals, and removed once they leave the visible game area. These systems work together to form the core challenge mechanics of the Flying Bird game.

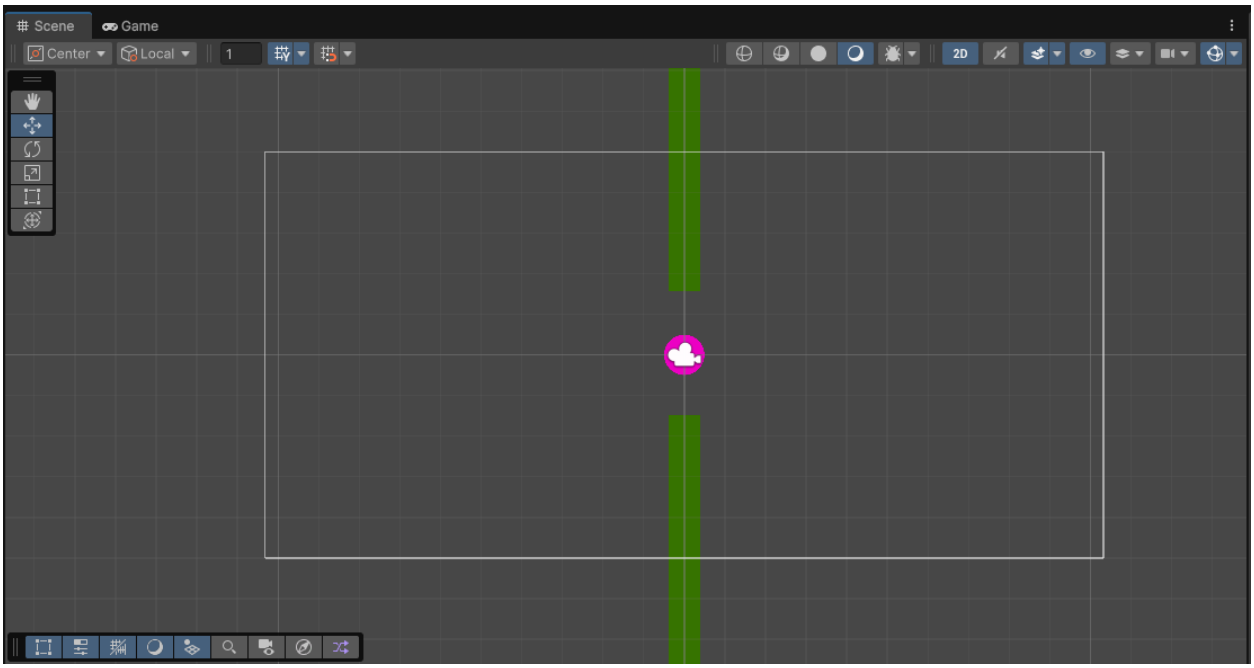
10.2- Pipe Sprite

The pipe obstacle is constructed using Unity's built-in 2D sprites. An empty GameObject is first added to the scene and named "Pipe Obstacle." This GameObject serves as a container that groups the visual and collision elements of the obstacle into a single unit.

The Pipe Obstacle is composed of two sprites: a top pipe and a bottom pipe. A Rectangle 2D Sprite is added as a child of the Pipe Obstacle and named "Top Pipe". A Box Collider 2D component is added to this sprite so it can detect collisions with the bird. Unlike the bird, the pipe obstacle is not affected by gravity, so a Rigidbody2D component is not required.



After the Top Pipe is configured, it is duplicated to create a second child GameObject named “Bottom Pipe.” The Bottom Pipe’s position is adjusted so it appears as a mirrored version of the top pipe, with a vertical gap between them. This gap is intentionally left open so the bird can fly through it during gameplay.



At this stage, the Hierarchy should show a single parent GameObject named “Pipe Obstacle” with two child GameObjects named “Top Pipe” and “Bottom Pipe.”

10.3 – Pipe Movement Scripting

To make the pipes move across the screen, a new C# script is added to the Pipe Obstacle GameObject. With Pipe Obstacle selected, a new script named “PipeMove” is created and attached. This script controls horizontal movement from right to left.

A public floating-point variable named moveSpeed is added and set to a value of 5. Unity positions objects using a three-value vector (x,y,z), even in 2D games, because Unity operates on a 3D coordinate system internally. Since only horizontal movement is required, the script modifies the x-axis using Vector3.left, which represent the vector (-1,0,0).

Initially, the pipe position is updated every frame by adding this movement vector to its current position.

```
public class PipeMove : MonoBehaviour

{
    public float moveSpeed=5;
    void Start()
    {

    }
    void Update()
    {
        transform.position = transform.position + (Vector3.left * moveSpeed);

    }
```

```
}
```

When this code is tested, the pipe moves extremely fast because the movement is tied directly to the frame rate. On systems running at very high frames per second, this results in inconsistent gameplay. To correct this, the movement calculations is multiplied by Time.deltaTime, which represents the amount of time that has passed since the previous frame. This ensures consistent movement regardless of hardware performance.

```
public class classPipeMoveScript : MonoBehaviour

{
    public float moveSpeed=5;
    void Start()
    {

    }
    void Update()
    {
        transform.position = transform.position + (Vector3.left *
moveSpeed)*Time.deltaTime;

    }
}
```

10.4— Pipe Move Deadzone Scripting

As pipes move offscreen to the left, they continue to exist in memory unless explicitly removed. To prevent unnecessary memory usage, a “dead zone” is implemented.

A float variable named “deadZone” is added to the PipeMove script. This value represents an X position beyond the left edge of the screen. When a pipe’s position moves past this value, it is destroyed using Unity’s built in Destroy() function.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PipeMove : MonoBehaviour

{
    public float moveSpeed=5;
    public float deadZone = -15;
    void Start()
    {

    }
    void Update()
    {
        transform.position = transform.position + (Vector3.left *
moveSpeed)*Time.deltaTime;

        if(transform.position.x<deadZone)
        {
```

```
            Destroy(gameObject);
        }
    }
}
```

10.5 – Pipe Prefab Creation

Once the Pipe Obstacle is fully configured and moving correctly, it is converted into a prefab. This is done by dragging the Pipe Obstacle GameObject from the Hierarchy panel into the Asset folder. Prefabs allow identical copies of an object to be created.

A successful prefab conversion is indicated by a blue cube icon appearing next to the object in the Hierarchy panel instead of a gray cube outline. After the prefab was created, the Pipe Obstacle GameObject can be deleted from the Hierarchy panel.

10.6—Completed Scripts

PipeMove Script:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PipeMove : MonoBehaviour
{
    public float moveSpeed=5;
    public float deadZone = -15;

    void Start()
    {

    }

    void Update()
    {
        transform.position = transform.position + (Vector3.left *
moveSpeed)*Time.deltaTime;

        if(transform.position.x<deadZone)
        {
            Destroy(gameObject);
        }
    }
}
```

10.7—Exercises

Reinforcement

The reinforcement section is used to check the understanding of the information that was presented in the PowerPoint/document. Please answer the questions in your own words.

R—10.1 Why is the purpose of grouping GameObjects under a single parent GameObject?

R—10.2 What is the purpose of a Box Collider 2D and a Rigidbody2d components?

R—10.3 Why is Vector3.left used instead of manually writing a vector like (-1, 0, 0)?

R—10.4 How does Time.deltaTime ensure consistent movement across different hardware?

R—10.5 What would happen if Time.deltaTime were accidentaly removed from the movement calculation?

R—10.6 What is the purpose of the Destroy() function?

R—10.7 How can you visually confirm that a GameObject has been converted into a prefab?

Project

The project section's tasks are used in work toward creating the final Flying Bird game project for the end of the course.

P—10.1 Create an GameObject named Pipe Obstacle using Unity's built-in 2D sprites

P—10.2 Create and attach a C# script that moves Pipe Obstacle across the screen from the right to the left and destroys the Pipe Obstacle when it's offscreen.

P—10.3 Convert Pipe Obstacle GameObject into a resuable prefab.